

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problems Mailbox.**

```

        ASSERT( 0 )!
        brc = FALSE;
        break;
    }

    // Now, save site page include-exclude list in the placement_sites table
    // pos = targetPages.GetStartPosition();
    //DWORD dwPageID;
    while (pos)

    {
        targetPages.GetNextAssoc( pos, dwPageID, blank );
        wprintf( buf, "Insert Placement_PagedSet Id_page_id,include values(%d,%d,%d",
                dwPageID, dwPageID, includepage );
        wprintf( buf, "\n" );
        if (lafmain.exec( buf ) != -1)
            addID( dwPageID, includepage );
        else
            break;

        ASSERT( 0 )!
        brc = FALSE;
        break;
    }

    if (lafmain.exec( buf ) != -1)
        wprintf( buf, "\n" );
    else
        break;

    // Delete the site include-exclude list from the placement_sites table
    wprintf( buf, "Delete placement_intereste where ad_id=id, id %d\n",
            id );
    if (lafmain.execError( buf ) != 0)
        ASSERT( 0 )!
        brc = FALSE;
        break;

    // Delete the site include-exclude list from the placement_sites table
    wprintf( buf, "Delete placement_pages where ad_id=id, id %d\n",
            id );
    if (lafmain.execError( buf ) != 0)
        ASSERT( 0 )!
        brc = FALSE;
        break;

    // Delete the placement_intereste table
    wprintf( buf, "Delete placement_intereste where ad_id=id, id %d\n",
            id );
    if (lafmain.execError( buf ) != 0)
        ASSERT( 0 )!
        brc = FALSE;
        break;

    // Delete the placement_pages table
    wprintf( buf, "Delete placement_pages where id = %d, id %d\n",
            id, id );
    if (lafmain.execError( buf ) != 0)
        ASSERT( 0 )!
        brc = FALSE;
        break;

    // Delete the placement_intereste table
    wprintf( buf, "Delete placement_intereste where ad_id=id, id %d\n",
            id );
    if (lafmain.execError( buf ) != 0)
        ASSERT( 0 )!
        brc = FALSE;
        break;

    // Delete the user interests from the placement_intereste table
}

```

DC 069520

HIGHLY
CONFIDENTIAL

```

void Ad::Report()
{
    dayOfWeek = 0x7f;
    flag = production | spreadIndustry;
    frequency = 0;
    impressions = 0;
    type = Normal;
    domainType = 0;
    gender = 0;
    placement = 0;
    maxDuration = Empty();
    startTime = 0;
    endTime = 0;
    on = DefaultMask;
    browser = DefaultMask;
    domainType = DefaultMask;
    isp = DefaultMask;
    hourOfDay = DefaultMask;
    employees = DefaultMask;
    salesVolume = DefaultMask;
    gender = DefaultMask;
    includePage = 0;
    includesites = 0;
}

```

seriesCount = 0;
delete [] sICodes;
sICodes = NULL;
delete [] locations;
locations = 0;
locations = NULL;
targetPage.RemoveAll();
targetSite.RemoveAll();
siteCategories.RemoveAll();
interests.RemoveAll();
adescription.Copy();
titles.Copy();
jumpto.Copy();
}

Endit

DC 069521

HIGHLY
CONFIDENTIAL

GOOL AD BOOK | DMDP advertisement

```
// Get the ID of the newly added ad  
int adID = _adManager.AddAd(ad);
```

```

char strmle10_l1
    bind SQL_LONG lndD, 9
    select but, "select max(lnd) from placements"
    execute but
    fetchbut
    lmain.commit

    // If this is a barter ad, set max_impressions = to 1
    if (type == Barter)
        ASSERT(0 != 1)
        return FALSE;

maxImpressions = 1

strcpy(but, "insert placements(jumpTo, max_impressions, type, ps_browser, domainType, ip, freq,
    max_amount, po_number, gender, active, approved, fileName);");
if (insertion)
    strcat(but, "start_time");
    strcat(but, "end_time");
street(but, "1 values(1);

addvalue(but, jumpTo);
addvalue(but, maxImpressions);
addvalue(but, type);
addvalue(but, on);
addvalue(but, browser);
addvalue(but, domainType);
addvalue(but, leap);
addvalue(but, frequency);
addvalue(but, imgSeries);
addvalue(but, description);
addvalue(but, hourOfDay);
addvalue(but, dayOfWeek);
addvalue(but, employees);
addvalue(but, salesVolume);
addvalue(but, maxAmount);
addvalue(but, responsible);
addvalue(but, gender);
addvalue(but, active);
addvalue(but, approved);
addvalue(but, fileName, FALSE);

if (targeted)
    strcat(but, "Ad.Targeted");

strcpy(but, "update placements set = 1
    char but[1024];
    char strmle10_l1;
    strcpy(but, "max_impressions = 1
    // Don't update max_impressions if this is a barter ad. REP.RET.
    // credits the placement so we don't want to overwrite the
    // barter credits
    if (type == Barter)
        street(but, "max_impressions");
    else
        addvalue(but, jumpTo);
        addvalue(but, type);
        addvalue(but, on);
        addvalue(but, browser);
        addvalue(but, domainType);
        addvalue(but, leap);
        addvalue(but, frequency);
        addvalue(but, imgSeries);
        addvalue(but, description);
        addvalue(but, hourOfDay);
        addvalue(but, dayOfWeek);
        addvalue(but, employees);
        addvalue(but, salesVolume);
        addvalue(but, description);
        addvalue(but, maxAmount);
        addvalue(but, responsible);
        addvalue(but, gender);
        addvalue(but, active);
        addvalue(but, approved);
        addvalue(but, fileName);

street(but, "1 values(1);

if (targeted)
    strcat(but, "Ad.Targeted");

strcpy(but, "select max(lnd) from placements");
execute but;
fetchbut;
lmain.commit

```

```
settime( stime, 9, "m/d/y", gmtime( &currentTime ) );
```

```
addvalue( buf, stime );
```

```
else
```

```
{ streetbuf = (NULL); }
```

```
streetbuf = "end_time";
```

```
if (endtime)
```

```
settime( stime, 9, "m/d/y", gmtime( &endtime ) );
```

```
addvalue( buf, stime, FALSE );
```

```
else
```

```
{ streetbuf = "null"; }
```

```
streetbuf = "where_id";
```

```
addvalue( buf, id, FALSE );
```

```
if (listmain.exists( buf ) != 1)
```

```
{ ASSERT( 0 );
```

```
return( FALSE );
```

```
return( AddPlacementTable( id ) );
```

```
return( FALSE );
```

```
char buf[1024];
```

```
BOOL brc = TRUE;
```

```
while (TRUE)
```

```
{
```

```
////////////////////////////////////////////////////////////////////////
```

```
// Now save the locations to the "placement_locations" table
```

```
////////////////////////////////////////////////////////////////////////
```

```
POSITION pos = siteCategories.GetStartPosition();
```

```
DWORD dwInterestID;
```

```
BOOL bJunk;
```

```
while (pos)
```

```
{
```

```
siteCategories.GetNextAssoc( pos, dwInterestID, bJunk );
```

```
wprintf( buf, "insert placement_sitecategory_id,interest_id values(%d,%d)",
```

```
dwInterestID, interestID );
```

```
if (listmain.exists( buf ) != 1)
```

```
{ ASSERT( 0 );
```

```
brc = FALSE;
```

```
break;
```

```
}
```

```
////////////////////////////////////////////////////////////////////////
```

```
// Now save the user interests to the placement_interest table
```

```
////////////////////////////////////////////////////////////////////////
```

```
pos = interests.GetStartPosition();
```

```
while (pos)
```

```
{
```

```
interests.GetNextAssoc( pos, dwInterestID, bJunk );
```

```
wprintf( buf, "insert placement_interestcategory_id,interest_id values(%d,%d)",
```

```
dwInterestID, interestID );
```

```
if (listmain.exists( buf ) != 1)
```

```
{ ASSERT( 0 );
```

```
brc = FALSE;
```

```
break;
```

```
}
```

```
////////////////////////////////////////////////////////////////////////
```

```
// Now save the include-exclude list in the placement_sites table
```

```
////////////////////////////////////////////////////////////////////////
```

```
pos = targetSites.GetStartPosition();
```

```
DWORD dwSiteID;
```

```
while (pos)
```

```
{
```

```
targetSites.GetNextAssoc( pos, dwSiteID, bJunk );
```

```
wprintf( buf, "insert placement_sitecategory_id,site_id,include values(%d,%d,%d)",
```

```
dwSiteID, siteID, include );
```

```
if (listmain.exists( buf ) != 1)
```

```
{ ASSERT( 0 );
```

DC 069519

CONFIDENTIAL
HIGHLY

// sitepage.cpp

```

include "siteUtil.h"
include "object.h"
include "dictObject/db.h"
include "dictObject/objUtil.h"
include "dictObject/dbUtil.h"

void message(const char *p)
{
    sitepage::SitePageGet()
}

id = 0
siteID = 0
categorized = FALSE;

void SitePage::loadCategories()
{
    DWORD interestID;
}

Cursor C;
C.bind(SQLC_LONG, interestID, siteID(interestID));
char sql[1024] = "select interest_id from page_categories where page_id=?";
addValue(sql, id, FALSE);
strcat(sql, " union all select interest_id from site_categories where site_id=?");
addValue(sql, siteID, FALSE);

C.executeSQL();
while (C.fetchNext()) {
    categories.Add(interestID);
}
}

extern BOOL defaultAddMode;
sitePage::SitePage::lookUpPage(Database db, const char *from, const char *requestName)
{
    // from key format: sitekey/docname
    if (from == 0)
        return 0;

    if (strlen(from) == 0)
        return 0;

    if (strlen(from, "www.", 4) == 0)
        from = "www";

    const char *q = strchr(from, '/');
    if (q == 0 || strlen(q) > 75)
        return 0;

    const char *key = q + 1;
    // truncate a unique number from the end of the key
    const char *lastSlash = strrchr(key, '/');
    if (lastSlash != key) {
        key = key - strlen(key);
    }
    else
        key = from;

    if (key[0] == '#')
        key = key + 1;
    // truncate to column width
}

SitePage * new SitePage()
{
    Cursor c(db);
    C.bind(SQLC_LONG, id, siteID, 4);
    C.bind(SQLC_LONG, sp_id, siteID, 4);
    C.bind(SQLC_LONG, sp_approved, siteID, 4);
    char sql[1024] = "select id,site_id,sp_id,sp_approved from sites where keyname=?";
    addValue(sql, key, FALSE);
    C.executeSQL();
    if (C.fetchNext()) {
        id = C.get("id");
        siteID = C.get("site_id");
        sp_id = C.get("sp_id");
        sp_approved = C.get("sp_approved");
    }
    return p;
}

void SitePage::add(Database db, const char *keyname)
{
    char buf[152] = "insert sitepages(junk, keyname, site, categorized) values('','')";
    addValue(buf, keyname);
    addValue(buf, siteID);
    addValue(buf, 0);
    categorized = FALSE;
    strcat(buf, "\n");
    if (db.execute(buf) != 1) {
        TPEC(error, adding sitekey\n");
        Cstring e = "sql: ";
        e += buf;
        ASSERT(FALSE);
        TPEC();
        message(e);
    }
}

Cursor C(db);
id = 0;
C.bind(SQLC_LONG, id, 4);
C.bind("id", "select id from sitepages where keyname=?");
C.bind("site_id", "select id,site_id,sp_id,sp_approved from sites where keyname=?");
C.bind("sp_id", "select id,sp_id,sp_approved from sites where keyname=?");
C.bind("sp_approved", "select id,sp_id,sp_approved from sites where keyname=?");
if (C.fetchNext()) {
    id = C.get("id");
    siteID = C.get("site_id");
    sp_id = C.get("sp_id");
    sp_approved = C.get("sp_approved");
}
C.fetchNext();
if (C.fetchNext() != 1)
    return p;
return p;
}

```

// Didn't find the page. Add page if site is correct.

```

Cstring siteKey(from, q + (from));
int approved = 0;
Cursor C(db);
C.bind(SQLC_LONG, sp_siteID, siteID(sp_siteID));
C.bind(SQLC_LONG, sp_approved, siteID(sp_approved));
char sql[1024] = "select id,approved from sites where keyname=?";
sql += siteKey + "\n";
executeSQL();
if (C.fetchNext() != 0) {
    if (C.get("approved") == 0) {
        message(Cstring("unapproved site: ") + (from));
    }
    else {
        p->add(db, key);
    }
}
else {
    delete p;
}
p = 0;
if (defaultAddMode) {
    if (message(Cstring("unknown site: ") + (from))) {
        message(Cstring("unknown site: ") + (from));
    }
}
}
}

```

CONFIDENTIAL

HIGHLY

DC 069516


```

// user.cpp

#include "stdln.h"
#include "object.h"
#include "d/cookie/db.h"
#include "d/cookie/lat_util.h"
#include "d/cookie/dbutil.h"

User User::lookupUserByIp(DWORD userId)
{
    User *u = new User();
    return u;
}

User User::lookupUserByAddress(DWORD ip)
{
    DWORD userId = networkNodeTable.getuserId(ip, FALSE);

    if(userId == 0) {
        // Try to get domain info at least. Note: If user is uniquely
        // identifiable, derive date process will create a record for the
        // user as soon as it gets a chance.
        userId = networkNodeTable.getuserIdJustNetworkNumber(ip), TRUE;
    }

    if(userId) {
        User u = lookupUserByUserId(userId);
        return u;
    }
}

class UserCursor : public Cursor
{
public:
    UserCursor(Database db, User *u) : Cursor(db),
        u(u) {}

    // Just gets field that aren't derivable from request header
    void minimalBind()
    {
        bind(SOL_C_LONG, u->ipPriority, &isSet(BOOL));
        bind(SOL_C_LONG, u->hasCookie, &isSet(BOOL));
    }

    User *u;
};

void UserCursor::lookupPharmacyInfo(Database db)
{
    if(userId == 0) {
        return;
    }

    Cursor c(db);
    char sql[128];
    sprintf(sql, "select email from users where id=%d", userId);
    c.bindNull();
    connect(c);
    fetchNext();
    db.commit();
}

void User::lookupUserByIp(Database db, DWORD userId, BOOL *timedOut)
{
    User *u = new User();
    UserCursor c(db, u);
    char sql[128];
    sprintf(sql, "select ipPriority, hasCookie from users where id=%d", userId);
    if(timeout != 0)
        c.setTimeout(timeout);
    c.execute();
}

```

DC 069514

CONFIDENTIAL
HIGHLY

```

    if(*timedOut) {
        *timedOut = TRUE;
        delete u;
        u = 0;
    }
    else if(c.fetchNext()) {
        u->userId = u->userId;
        u->ipPriority = c.bind(SOL_C_LONG, u->userId, 4);
        char sql[128];
        sprintf(sql, "select ipPriority, hasCookie, id from users where ip=%d",
            ip);
        if(timeout != 0) char sql[128];
        if(timeout != 0) {
            C.setTimeOut(timeout);
            C.executeQuery();
        }
        if(c.timeOut()) {
            *timedOut = TRUE;
            delete u;
            u = 0;
        }
        else if(c.fetchNext()) {
            u->ipPriority = c.bind(SOL_C_LONG, u->userId, 4);
            ASSERT(c.fetchObject());
            ASSERT(FALSE);
            return;
        }
        char buf[128];
        sprintf(buf, "update users set ipPriority=%d where id=%d",
            ip);
        db.executeUpdate(buf);
        db.commit();
    }
}

void User::makePerformance(Database db)
{
    if(!stepUserObject())
        return;

    ASSERT(name.length() > title.length() && email.length() > name.length());
    ASSERT(proxy.length() > title.length());
    ASSERT(domainType.length() > proxy.length());
    ASSERT(ip.length() > domainType.length());
    ASSERT(cookie.length() > ip.length());
    ASSERT(cookie.length() > domainType.length());
    ASSERT(cookie.length() > proxy.length());
    ASSERT(cookie.length() > ip.length());
    ASSERT(cookie.length() > title.length());
    ASSERT(cookie.length() > name.length());
}

// add to DB
char buildSql()
{
    "insert users (in_browser,bver1,bver2,os,domain_type,in_proxy,in_networkdesc,ip_priority,
    strctbou, *values ";
    addValue(bou, ip);
    addValue(bou, ip);
    addValue(bou, browser);
    addValue(bou, bver1);
    addValue(bou, bver1);
    addValue(bou, os);
    addValue(bou, proxy);
    addBool(bou, isNetworkDescription);
    addBool(bou, (ipPriority & 0x000000ff));
    addBool(bou, (ipPriority & 0x0000ff00));
}

```

```
addBool(but, hasCookie, FALSE);
assert(but, !p);

if( db.beginTransaction(but) == 1 ) {
    Cursor c(db);
    c.bind(TOL_C_LONG, userId, 4);
    c.setString(but, "select max(id) from users where ip='");
    c.addValue(but, ip, FALSE);
    c.exec(but);
    c.fetchNext();
    ASSERT(userId != 0);
}

db.commit();
```

DC 069515

CONFIDENTIAL
HIGHLY

41

18

```
if (n > 1) {
    buf[0] = '0';
    TRACK("0", buf);
}
```

```
else
{
    GetRequest gr(c, v, r, front);
    selectRequest gr(c, v, r, front);
}
```

```
sendit
gr.service();
}
```

```
listener *listener = 0;
```

```
c->nthread = 0;
int maxThreads = 1;
```

```
JINT ListenerThreadUp(void)
```

```
static DWORD ad = GetTickCount();
sendit ad;
}
```

```
while (1) {
    socketd_in from;
    Connection *c = listener->WaitForConnection(front);
    if (c) {
        if (defined("PORT"))
            int port = _PORT;
        else
            int port = 80;
        sendit
        Listener *newListener = new Listener(port);
        if (newListener)
            if (newListener->listen())
                if (defined("ANSWER"))
                    errlog.open("connection/errlog.txt",
                                fion_out | fosi_app,
                                flibutish_Zead);
                ASSERT(errlog.isOpen());
                errlog.flush();
            else
                error("..... Ad server started\n");
        }
    }
}
```

```
if (defined("ADSVR"))
    if (defined("OPENTABLES"))
        sleep(100); // Idiot this is a test; sometimes it doesn't listen right. Just a hunch
    else
        ASSERT(FALSE);
return TRUE;
}
```

```
bool startServer()
{
    if (defined("ADSVR"))
        if (defined("OPENTABLES"))
            AddressMessage("Error opening tables!");
    return FALSE;
}
```

```
if (!initTables())
    return FALSE;
}
```

```
if (!initCount())
    initCount();
}
```

```
if (defined("PORT"))
    if (defined("OPENTABLES"))
        AddressMessage("Error opening tables!");

```

```
else
    ASSERT(FALSE);
return TRUE;
}
```

```
if (defined("PORT"))
    if (defined("OPENTABLES"))
        AddressMessage("Error opening tables!");

```

```
else
    ASSERT(FALSE);
return TRUE;
}
```

```
if (defined("PORT"))
    if (defined("OPENTABLES"))
        AddressMessage("Error opening tables!");

```

```
else
    ASSERT(FALSE);
return TRUE;
}
```

```
if (defined("PORT"))
    if (defined("OPENTABLES"))
        AddressMessage("Error opening tables!");

```

```
else
    ASSERT(FALSE);
return TRUE;
}
```

```
if (defined("PORT"))
    if (defined("OPENTABLES"))
        AddressMessage("Error opening tables!");

```

```
else
    ASSERT(FALSE);
return TRUE;
}
```

```
if (defined("PORT"))
    if (defined("OPENTABLES"))
        AddressMessage("Error opening tables!");

```

```
else
    ASSERT(FALSE);
return TRUE;
}
```

```
if (defined("PORT"))
    if (defined("OPENTABLES"))
        AddressMessage("Error opening tables!");

```

```
else
    ASSERT(FALSE);
return TRUE;
}
```

```
if (defined("PORT"))
    if (defined("OPENTABLES"))
        AddressMessage("Error opening tables!");

```

```
else
    ASSERT(FALSE);
return TRUE;
}
```

```
// TEMP
connection c;
if (c.connect("www.microsoft.com", 80) != c.write("GET /ad HTTP/1.0\r\n\r\n", 22))
    while (1)
        char buf[256];
        int n = c.read(buf, 255);
    }
}
```

DC 069513

HIGHLY
CONFIDENTIAL

19-Jan-1996 10:13

四百九

18-Jan-1998 10:15

四百九

```
19-Jan-1996 10:15
SOLDs.cpp
SELECT * FROM placement WHERE ad_id = ?
    adValueInt, ad_id, FALSE);
    c.fetchNext();
    if (c.eof()) {
        if (n == 0) {
            // to do: count the # of sites first, and allocate that number
            // rather than 50
            n = 50;
            newSiteCodes = n;
        }
        ASSERT( ic.fetchNext() );
        adCode = n;
        break;
    }
}

// load regional
for (i = 0; i < ad.GetSize(); i++) {
    Region r;
    Ad ad = ad.GetAt(i);
    if (ad.IsTargeted())
        continue;
    if (n == 0)
        continue;
    if (n > 100)
        message("100 locations targeted");
    int n = 0;

    Cursor c;
    WORD country;
    CString state, sIp;
    int areaCode;
    char eq1[512] = "select count(*) from placement_locations where ad_id = ? ";
    adValueInt, ad_id, FALSE);
    c.fetchNext();
    if (c.eof()) {
        if (n == 0) {
            if (i == 0) {
                newRegion(n);
                adLocations[i];
            }
            country = country;
            state = state;
            sIp = sIp;
            areaCode = areaCode;
            adLocations[n] = n;
            ASSERT( ic.fetchNext() );
            break;
        }
        areaCode = 0;
    }
}
}

label.commit();

```

```
19-Jan-1996 10:15
SOLDs.cpp
if (ad.GetSize() == 0 || !targeting) {
    // db connection down, use some default ad
    makeDefault(ad);
}
if (defaultAd == 0) {
    TRACE("no default ad\n");
    message("no default ad");
}
return ad.GetSize() == 0 || defaultAd != 0;
```

DC 069511

HIGHLY
CONFIDENTIAL

```

#include "adbase.h"
#include "astream.h"
#include "object.h"
#include "dtoolkit/db.h"
#include "dtoolkit/dutil.h"
#include "dtoolkit/dpool.h"
#include "dtoolkit/crit.h"

// this ad displayed if a bad slotkey is encountered
const cSlotkeyAd = 49;

extern CriticalSection (fast)

Database ltmalm;
void message(const char *s)
BOOL defaultMode = FALSE;
static int ucold;
static void localstruct(item_t & t)
{
    t = ucold;
}

// This is temporary. Used for non-unique users.
// Eventually will be smarter about what to send to
// these users.
Ad *defaultAd = 0;
Ad *badkeyErrorAd = 0;

typedef CurrentAd * Ad ** AdArray

SOOL loadAdArrays(Ad **,
    DWord adverbAdID, // 0=all
    DWord fortargeting, // if fortargeting,
    // else exclusions
    BOOL activeOnly, // active=1 only
    BOOL includeExpired, // include where active=0
    DWord newerFirst, // order from newest
    DWord slotID = 0);

SOOL openSOOL()
{
    if (main.open() != 0)
        return FALSE;
    if (!openSOOL())
        return FALSE;
    if (!ltsOpen().Open(0, "ODCDSM1\\LUD-01\\PDB-", 1))
        return FALSE;
    if (!loadAdArrays(0, TRUE, TRUE, FALSE, FALSE))
        return FALSE;
    return TRUE;
}

```

DC 069508

**HIGHLY
CONFIDENTIAL**

```

static void makeDefaultAds(Addresss ads)
{
    if(strm.defaultPath("\\\\lan\\default_ad\\text"))
    {
        if(dbConn.isOpen())
        {
            ASSERT(FALSE);
            return;
        }
    }

    message("db connection failed, using default_ad\\text");

    defaultAdmed = TRUE;

    while(1)
    {
        char thizAd[128];
        char jump[128];
        int i=0;

        defaultAd = (n > jump);

        if(i == 0)
            break;

        Ad ad = *(new Ad);

        defaultAd = !ad.advertiserId;

        time_t now = time(&now) - 60 * 60 * 24 * 15;

        ad.startTime = ad.endTime = now + 60 * 60 * 24 * 15;

        ad.endtime = now + 60 * 60 * 24 * 15;

        ad.jumpTo = jump;

        ad.approved = 1;

        ads.AddAd(ad);

        // calc time zone adjustment
        CTime t = CTime::GetCurrentTime();
        tm gmt, local;
        t.GetGmtTimegmt();
        t.GetLocalTime(local);
        t.gmt.tm_hour += 24;
        uctole = (gmt.tm_hour - local.tm_hour) + 60 * 60;

        ads.setAdId(64);

        BOOL active = 1;
        getContigual("active", active);

        Adcursor rc;
        char sql[200];
        "select id,type,os,browser,domainType,ip,filename,jumpTo,frequency,Impression,active,\n"
        "n_impressions,show,adiffid,fee, '1/1/70', end,fee,\n"
        "flag, heure_du_jour, date_of_web_employee,sales,active,description,man_amount,po_number,\n"
        "approved,n_jumps,from_placement",\n
        "BOOL where = FALSE;

        if(!includedexpired) // (n_impressions<0 or n_shown<n_impressions) and \
        !(end_time==NULL or end_time->getdate()),\n
        where = TRUE;

        if(lectivity)
        {
            if(where)
                direct(sql, " and");
            else
        }
```

DC 069506

HIGHLY
CONFIDENTIAL

```
old Request::service()
{
    const char *p = strchr(request, '/');
    if (p)
        filename = CString(request, p - request);
    else
        filename = request;

    if (*p == '/') {
        // send default file
        //sendfile(hh,"my documents\internet address finder\iarmain.htm");
        if (defined(hh))
            sendfile("c:\inet\html\iarmain.htm");
        return;
    }

    else {
        if (defined(hh))
            CString(f, "c:\inet\html\");

        if (defined(hh))
            CString(f, "c:\inet\manage\");

        if (defined(hh))
            CString(f, "c:\inet\federation\");

        ASSERT(FALSE);
        CString(f, "johidf");
        //CString(f, "c:\inet\my documents\ad federation\");

        if (defined(hh))
            sendfile(f);
        return;
    }
}

void Request::sendInternalError()
{
    senderror("500 Internal Server Error");
}
```

```

// rememberd.cpp

// rememberd.h
#include <stdfa.h>
#include <object.h>
#include "rememberd.h"
#include </d/tool/hashuf.h>
#include </d/tool/crit.h>

// DWORD hash(const char *from, User *u)
//{
//    char buf[10];
//    wprintf("%s", u->name);
//    wprintf("%s", u->getID());
//    CSRing a = buf;
//    a += from;
//    return hashuf(a);
//}

```

```

// this is a test
{
    static int err;
    define INCPUT ( ASSERT((err=0), err+1) )
    define OUTPUT ( ASSERT((err>0), err-1) )

    void message(const char *):
        extern CriticalSection (set)

        struct key
        {
            DWORD userId;
            DWORD fromHash;
        };

        BOOL operator==(const Keys h) const
        {
            return userId == h.userId && fromHash == h.fromHash;
        }

        void setUserId(User *u)
        {
            if (u->userId)
                u->userId = u->ip;
            else
                u->ip = u->userId;
        }

        void setFrom(const char *from)
        {
            if (fromHash == hash((from)))
                fromHash = hash((from));
        }

        uint HashKey(Keys key)
        {
            return key.userId ^ key.fromHash;
            // default identity hash - works for most primitive values
            // return (uint)(void *)key >> 4;
        }

        struct value
        {
            DWORD adsent;
            DWORD time;
        };
};

class Memory
{
public:
    Memory();
    ~Memory();
    sent initHashTable(size_t sz);
};

void remember(Keys h, DWORD adId)
{
    sent remember(Keys h, DWORD adId);
}

void rememberSent(Ad User *u, const char *fromHash)
{
    Crit Cilock();
    INCPUT
    Keys h;
    h.setUserId();
    h.setFrom((fromHash));
    h.setFrom((fromHash));
    memory.remember(h, adId);
    OUTPUT
}

void rememberSent(Ad User *u, const char *fromHash)
{
    Crit Cilock();
    INCPUT
    Keys h;
    h.setUserId();
    h.setFrom((fromHash));
    memory.lookup(h);
    OUTPUT
}

```

```

        DWORD addent;
        DWORD time;
    }

    class Memory
    {
    public:
        Memory() : sent(100)
        {
            sent.initHashTable(sz);
        }

        void remember(Keys k, DWORD addt)
        {
            DWORD lookup(Keys k);
            k.setid(id);
            k.setfront(*comaddr);
            k.setfront(*comaddr);
            memory.remember(k, addt);
        }

        private:
        void pushget();
        CheapKey keys, value, values, sent;
        memory();
    };

    // INPUT
    // KEY k;
    // setid(id);
    // setfront(*comaddr);
    // memory.remember(k, addt);
    // OUTPUT
}

DWORD queryAddent(user *u, const char *comaddr)
{
    Crit critact;
    // INPUT
    KEY k;
    k.setid(id);
    k.setfront(*comaddr);
    k.setfront(*comaddr);
    DWORD d = memory.lookup(k);
    // OUTPUT
    return d;
}

```

HIGHLY
CONFIDENTIAL

DC:069507

HIGHLY

CONFIDENTIAL

KUTCH.CP

20-Jan-1996 18:13

1315

הנִזְקָנָה

```

// A truly random distribution is used for them rather than
// lottery
static int testCounter;
if (testCounter < 4 >> 0) { // Just try every 4 to save CPU
    // test ad availability
    lowestSI = 1051;
    int i = alert;
    while (i) {
        Ad ad = ads.get(i);
        if (ad.type == TestAd.adType & ad.si < lowestSI && ad.criteriolock(db, user)) {
            lowestSI = si;
            adlowestSI = ad.si;
            i = i - 1;
            break;
        }
        if (lowestSI <= 1050)
            return adlowestSI;
    }
}

lowestSI = siMax;
adlowestSI = defaultAd;

// Check elements first. This way we don't
// have to do ad matching for any targeted ads
// with high SI.
//
int i = start;
while (i) {
    Ad ad = ads.get(i);
    if (ad.adType == NormalAd.adType & ad.isTargeted() & ad.si < lowestSI &&
        lowestSI == ad.si) {
        lowestSI = ad.si;
        adlowestSI = ad.si;
    }
    if (i == start)
        break;
}

// this is temp, eventual all placements will have book rate
// you'll want to remove this to get better performance into ad matching
// it remains here worst case.
static int counter;
int i, counter, a, b;
{
    // for ad with no booking amount,
    // allow a targeted ad to run sometimes
    if (lowestSI == 100)
        lowestSI++;

    // for ads where we don't care about P impressions,
    // bias in favor of targeted
    if (lowestSI == 1100)
        lowestSI++;

    // do targeted
    // start
    while (i) {
        Ad ad = ads.get(i);
        if (ad.adType == NormalAd.adType & ad.isTargeted() &&
            ad.si < lowestSI && ad.preload(pager) &&
            ad.matches(user, pager) &&
            ad.enquiriesOK(db, user)) {
            if (found > goodOne)

```

```

// request.cpp

#include "adat.h"
#include "adtoolkit/adt.h"
#include "request.h"
#include "adtoolkit/adfutil.h"
#include "astream.h"

#ifndef _CONSOLE_
#define _CONSOLE_
#endif

extern ofstream cout;
void Impression();
const char *request;
const sockaddr_in from;
const ccl::request &request;
ccl::ccl;

Request::Request()
{
    Connection = "C";
    Verb = "GET";
    const char *request;
    const sockaddr_in from;
    userip = from.sin_addr.s_addr;
}

int spider = 0;

BOOL Request::sendfile(const char *filename, const char *insertstr)
{
    if (defined(_API))
        "outlog" << "end" << endl << filename << " " << inet_ntoa((in_addr) userip) << "\n";
    const char insertchar = '\0';
    BOOL lSpider = FALSE;
    CString hdr = "HTTP/1.0 200 OK\r\nContent-Type: ";
    if (strchr(filename, ".class") != 0)
    {
        hdr += "application/java\r\nContent-Length: ";
    }
    else if (strchr(filename, ".gif") != 0)
    {
        hdr += "image/gif\r\nContent-Length: ";
    }
    else
    {
        hdr += "text/html\r\nContent-Length: ";
    }
    if (defined(_API))
        Impression();
    sendfile(
        int gnt = 0;
        if (struktur(request, "Agent", "lycos") != 0)
        gnt = 1;
        if (struktur(request, "Intoset Robot") != 0)
        gnt = 2;
        if (struktur(request, "Agent", "WebCrawl") != 0)
        gnt = 3;
        if (gnt)
        lSpider = TRUE;
        spider = TRUE;
        cout << "Robot" << endl;
    }
}

const DWORDSIZ = 138000;
char temp[100];
Itoa(m, temp, 10); // content length
hdr = temp;
cout << "ContentLength" << endl;
if (v == GET || v == POST)
    CContent();
    cout << "/body\n";
    cout << endl;
return TRUE;
}

```

```
// match.cpp
// Ad Matching!
```

```
include "etdata.h"
include "objecte.h"
include "/d/coolkit/dbutil.h"
```

```
extern Ad *defaultAd;
extern Ad *badKeyErrorAd;
```

```
extern int nextAd;
```

```
int nextAd()
```

```
// Returns TRUE if this location is in region.
```

```
BOOL Location::inRegion(Pregions region)
```

```
{ if(region.country == 0 && country != region.country)
```

```
return FALSE;
```

```
if(region.state.isEmpty() && strcomplete(region.state) == 0)
```

```
return FALSE;
```

```
if(region.zipcode.isEmpty() )
    return TRUE;
```

```
// Returns TRUE if this location is in region.
```

```
BOOL Ad::areaCodeIsInRegion(areasCode areaCode)
```

```
{ if(areaCode != 0 && areaCode == region.areaCode)
```

```
return FALSE;
```

```
if(region.state.isEmpty() && strcomplete(region.state) == 0)
```

```
return FALSE;
```

```
if(region.zipcode.isEmpty() )
    return TRUE;
```

```
// string myZip = zipcode.left(s); // extract zipcode for now
```

```
string regZip = region.zipcode.left(s);
```

```
string regZipEnd = region.zipcode.left(s);
```

```
if(regZipEnd.isEmpty() )
    return regZip == myZip;
```

```
return myZip == regZip || regZipEnd
```

```
== regZip || regZip == regZipEnd;
```

```
BOOL Ad::exposureOK(Database db, User *user)
```

```
{ Ad *ad = user->getAd();
    if(ad->isTargeted())
        return TRUE;
```

```
    char sql[1024];
    "insert exposures values(:";
    addValue(sql, "id", FALSE);
    addValue(sql, "ad_id", FALSE);
    addValue(sql, "user_id", FALSE);
    addValue(sql, "user_getId()", FALSE);
    db.executeQuery();
}
```

```
return TRUE;
```

```
    BOOL Ad::spreadOK(sitePage *sitePage)
{
    // Is start time met?
    if( !started ) {
        time_t now;
        if( time(&now) < starttime )
            return FALSE;
        started = TRUE;
    }
    // Impressions OK?
    if( nShown > maxImpressions || nShown == 1120 )
        return FALSE;
    if( impressionsLeft() < 1 )
        return FALSE;
    if( targetSites.isEmpty() )
        if( nIps == 0 )
            return FALSE;
        sitePage->setSiteID( v );
        sitePage->targetSitesLookup( sitePage->siteID, v );
        if( includesSite( v ) ) {
            // If we have page to target too, ok if site
            // doesn't match (check if page does next).
            if( found && targetPages.isEmpty() )
                return FALSE;
            else if( !found )
                return FALSE;
        }
    }
    return TRUE;
}
```

```
BOOL Ad::matchesUser(User *user, sitePage *sitePage)
{
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !includesPage( v ) )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
    if( targetPages.isEmpty() )
        if( altPage == 0 )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
    if( !found )
        if( !found )
            return FALSE;
}
```

```
char sql[1024];
db.executeUpdate();
}
```

DC 069502

HIGHLY
CONFIDENTIAL

માર્ગદરોહિની પત્રા .

page 311

10-Jan-1996 18:15

四百一

DC 069500

CONFIDENTIAL
HIGHLY

16-Jan-1998 18:10

Page 100

OBJECTS.CPP

```
sendit
{
    arrlog.Flush();
}

// temp, just return first ad (ISS)
// return new Ad(ads.ElementAt(0)),
// return new Ad( defaultAd ),
// return 0;
}

result
result
```

DC 069501

HIGHLY CONFIDENTIAL

```
// cookie.cpp
// ...
// include "stdapi.h"
// include "object.h"
// ...
// cookie
const Cookies Cookie::operator=(const char *a)
{
    delete(a, "X", avalue);
    return *this;
}

// static
Cookie::Cookie::alloc(DWORD user_id)
{
    ASSERT(user_id != 0);
    Cookie k;
    k.value = user_id;
    return k;
}

// Get value for a particular cookie name from the HTTP header
// hdr - points to the Cookie: field in the header
void Cookies::getFromHeader(const char *hdr, const char *name)
{
    char .. q; // skip "Cookie:"  

    const char *p = strchr(hdr, '\r');
    if(p) {
        lstring nm = name;
        nm .. .";";
        const char *q = strchr(hdr, nm);
        if(q < p) l(q - q - nm.GetLength());
        *this = q - nm.GetLength();
    }
}
```


-

DC 069499

**HIGHLY
CONFIDENTIAL**

```

// don't know location, except country
location.setCountry("US");
location.setCity("San Jose");
location.setZipCode("95134");
location.setStreetCode("123 Main St");

else {
    // lookup by cookie, cookie.value, timeout;
    u = _lookupUserByIp(db, cookie.value, timeout);
    if (u != NULL) {
        u->uniqueName = uniqueName;
        u->ip = ip;
    }
}

else {
    if (defaultAdMode) {
        // db conn down
        u = new User();
        u->uniqueName = uniqueName;
        u->ip = ip;
        u->cookie.value = cookie.value;
    }
    else {
        // Couldn't find user record, we will need to
        // assign a new cookie. Do not load by IP, because
        // we don't want this user sharing a record
        // with others without cookies.
        // Note: generally, this shouldn't happen.
        cookie.value = 0;
    }
}

else if (!timedOut) {
    u = _lookupUserByAddress(db, ip, timeout);
    if (u != NULL) {
        u->ip = ip;
        u->hasCookie = FALSE;
    }
}

if (u == 0) {
    // make a default user object
    u = new User();
    u->uniqueName = uniqueName;
    u->ip = ip;
    u->timedOut = _timedOut;
    u->hasCookie = FALSE;
}

u->headerDerivesFromRequestHdr;

if (!cookie.isNull()) {
    u->hasCookie = TRUE;
}

if (loadDemographics != _timedOut) {
    u->_getNetworkInfoLab(reftime ? reftime : u->timedOut - 0);
}

return u;
}

// get cookie for lookup

Cookie cookie;

const char *cCookie = strstr(requestHdr, "Cookie:");
if (cCookie != NULL) {
    cookie.getFromHeader(cCookie, "HTTP");
}

//.....
```